Hashmap WITSML Server Documentation

Release 0.0.1

Hashmap

Mar 27, 2019

Contents:

CHAPTER 1

Project Overview

These guides will help you get started as quickly as possible.

1.1 Solution Architecture

1.1.1 Introduction

What is Drillflow

Drillflow is an API facade that allows Oil and Gas software systems that currently expose drilling data to leverage WITSML to exchange data between other software systems and vendors. It is packaged as a docker image or a Spring Boot application to allow for minimum hassle at deploy time. It is also intended to be extensible and horizontally scalable to handle everything from a one time bulk load to a streaming application.

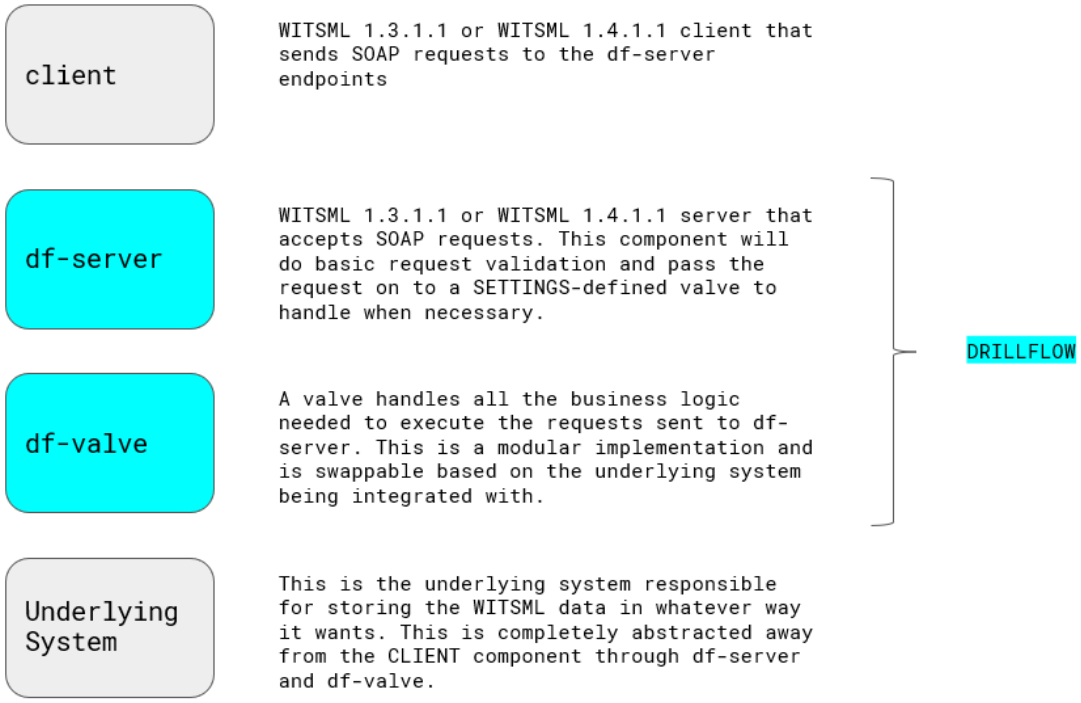
WHY?!?!?

WITSML servers have been implemented in many forms and fashions over the relatively long lifespan of WITSML. The point of Drillflow is to ease the burden for software developers and system integrators to make use of WITSML data. Our goal is that if we can get this hurdle out of the way quicker, time to value is reduced.

Our goal was to implement a WITSML server on a modern stack: Java 11, Spring Boot with CXF, deployed with docker.

So How does it Work

As stated in the "What is…" section above, Drillflow is an API facade. So what does this mean, exactly? Drillflow does not have any data persistance, or traditional transactional logic. What it does do is sit in front of a REST/Thrift/Protobuf/etc… API that will allow for data to be accessed via the WITSML data model and semantics.



1.1.2 Core Concepts

There are a few key concepts to understand in Drillflow. First are the major components:

- **df-server**: This is the WITSML SOAP API exposed via Tomcat and CXF, also the application server
- **df-valve**: This is the extensibility point that allows Drillflow to proxy WITSML commands to another system

The valve is further broken into two key pieces:

- **Translator**: The part of the valve that allows the actual body of the commands to be translated between df-server and the underlying system
- **Delegator**: The part of the valve that is responsible for actually executing the command on the underlying system and returning the response to the valve so that it can be returned to df-server

Let's dig into these a little further in the next session:

Server

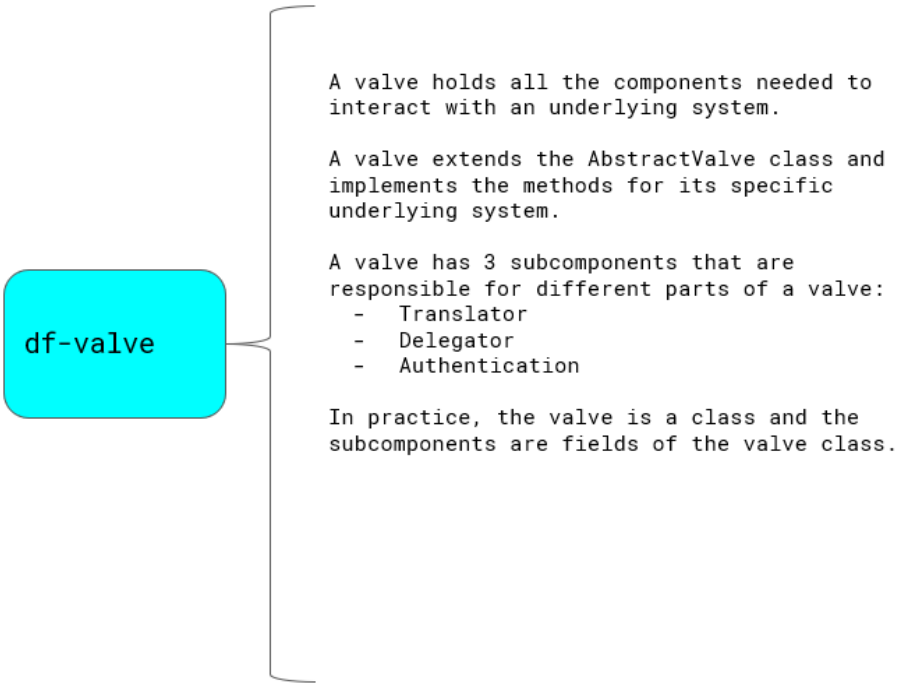
To be clear, there is really not much in this part that needs to be modified. Overall this solution is responsible for handling WITSML specific exchanges with a client. This, currently, can handle WITSML 1.4.1.1 and 1.3.1.1 schema versions and conforms to the 1.2.0 WSDL. Overall this portion of the server is directly responsible for the 3 required WITSML functions:

- GetBaseMsg
- GetVersion
- GetCap

Each one of these is defined mostly by properties files that can be modified at deploy time either by passing environement variables, passing a properties file, or Java system properties. Overall this is really managed by Spring so any of the options listed Here.

The one notable exception is GetCap. The contact information, server metadata (name, etc…) is from a properties file, however, the actual function capabilities are advertised to df-server via the Valve. There is a function that allows the valve to notify, df-server what functions, and what data object for each function is available. This is a key point, as this allows the Valve to have full-control over the objects and functions that it wants to support. This is not imposed on the underlying system by df-server. This leads us into our next discussion: The valve.

Valve



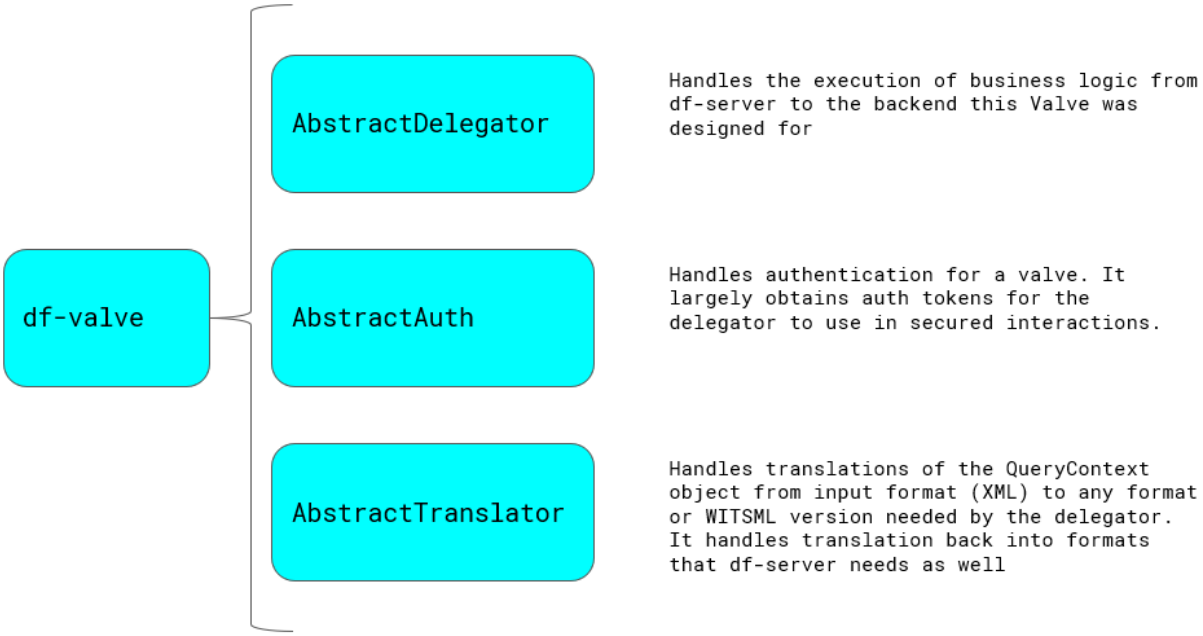
As seen in the image above, the valve is an encapsulation of all the elements required to communicate with an underlying system.

NOTE: There is one main thing that need to be noted about this… Class loader isolation… there is none. Therefore dependency conflicts are possible if care is not taken. This is something that might be addressed in the future, but at the moment that complexity is not warranted for this solution, yet.

The valve consists of three components:

- Authenticator
- Translator
- Delegator

These can be seen along with their descriptions in the image below:



As seen in the diagram above, each component of the valve is defined as an abstract class that can then be implemented by the valve implementation.

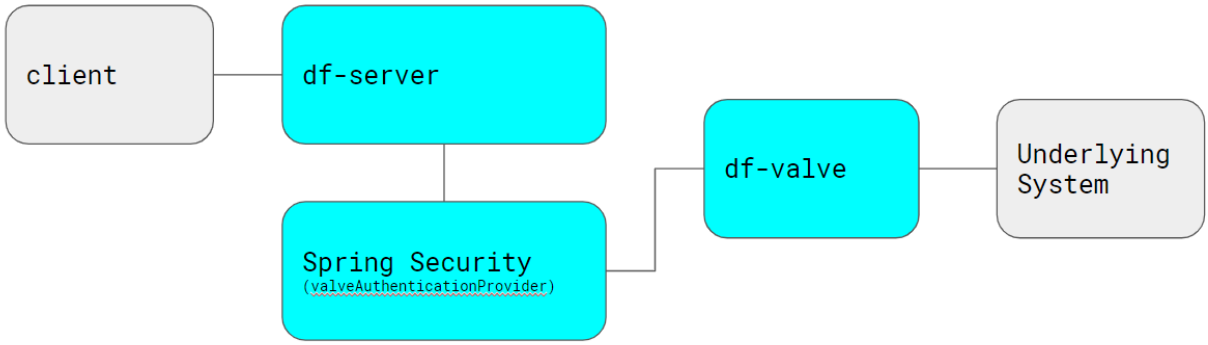
There is a distinct difference between the Translator/Delegator and the Authenticator. The Authenticator is used via a custom authentication provider, the ValveAuthenticationProvider. This is called during the authentication flow of Tomcat as a function is called. The main purpose of this design is to allow the underlying solution to implement whatever authentication it sees fit. There is a more detailed section on authentication patterns in a separate section located **here**.

The Translator and Delegator components are only invoked through an actual query chain initiated by a client calling a function of the WITSML API in **df-server**.

1.2 Authentication

NOTE: Drillflow handles Authentication NOT Authorization. Authorization is the responsibly of the underlying system, not Drillflow.

1.2.1 Introduction



As can be seen in the diagram above, **df-server** leverages Spring security to handle authentication. This is done via the ValveAuthenticationProvider. Spring Security is configured to use this authentication provider which calls the **authenticate()** method of IValve. The authenticate method is responsible for executing the necessary elements to authenticate with the underlying system.

1.2.2 WITSML Authentication

It needs to be noted at this point that version 1.3.1.1 and 1.4.1.1 (the currently supported versions of the API) only support HTTP BASIC authentication. This means that whatever authentication system that the underlying system has, must be able to handle a username and password combination. This poses issues with most modern solutions. In this section we will aim to propose some solutions to common integrations.

CHAPTER 2

Developer Guide

These guides will help you get started as quickly as possible.

2.1 Getting Started

2.1.1 Introduction

NOTE: These instructions are generic until you get to the running section in which they are specific to the DoT valve

Building Drillflow

Let's talk about building Drillflow. It is optimized to build on Java 8+ (including 11 for the latest LTS release). It is most thoroughly tested on OpenJDK but has been built and runs on the Oracle JDK as well. In order to build Drillflow you will need the JDK 8+ (preferrably 11), Maven 3.3+ and git to pull the code (or you can download the zip from the GitHub repo. Additionally to build the docker image you will need docker.

2.1.2 Pulling the Code

First we will start with cloning the repo... this is easy enough

```
git clone https://github.com/hashmapinc/Drillflow.git
```

This will clone Drillflow to your local repository

2.1.3 Building the Code

Building the code is also relatively simple thanks to the magic of Maven.

Change directories into Drillflow

```
cd Drillflow
```

And build the project

```
mvn clean package
```

You should end up with something like this:

```
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Summary for DrillFlow 0.0.1-SNAPSHOT:                    [INFO]
[INFO] DrillFlow ..........................................
SUCCESS [ 0.750 s]               [INFO] DrillFlow Valve Module ..............
.............. SUCCESS [ 46.487 s]                [INFO] DrillFlow Application
Module ....................... SUCCESS [ 12.460 s]                        [INFO]
------------------------------------------------------------------------
[INFO] BUILD SUCCESS[INFO] ------------------------------------------------------------------------
[INFO] Total time:  01:00 min    [INFO] Finished at:  2018-12-21T12:35:52-06:00
[INFO] ------------------------------------------------------------------------
```

2.1.4 Running Drillflow

Assuming no errors have occurred then you will have a ready to run application.

Now before you can get started you need to have some system behind Drillflow providing the data as described in the *solution architecture*

so you need to set 2 environment variables first: **VALVE_BASE_URL**: this sets the base URL of the REST queries to make on the backend **VALVE_API_KEY**: this sets the API key to use against the REST API on the back end

Change directories into the df-server/target directory:

```
cd df-server/target
```

And you can now execute:

```
java -jar df-server-0.0.1-SNAPSHOT.jar
```

At this point you will have a WITSML serving getCap, getVersion, getBaseMsg at the following url:

```
http://localhost:7070/Service/WMLS
```

The WSDL is available at:

```
http://localhost:7070/Service/WMLS?wsdl
```

2.1.5 Running the Docker Container

In the case where you don't want to build the code and just want to run the container it is as simple as:

Pulling the container:

```
docker pull hashmapinc/Drillflow:latest
```

Running the container:

```
docker run -p 7070:7070 -e VALVE_API_KEY='<api key>' -e VALVE_BASE_URL='<put
in your base url here>' hashmapinc/drillflow:latest
```

Replacing **<api key>** with the actual API key and **<put in your base url here>** with the actual base url.

Ideally this would be injected with a configuration management tool such as Consul.

CHAPTER 3

Integration Guide

These guides will help you integrate the WITSML Server API with your API implementation

3.1 Setting the WITSML Version

This guide documents how to set the WITSML version that the server reports as SUPPORTED by the WMLS_GetVersion API.

3.1.1 In Code

The WITSML version can be set in code by modifying the application.properties file. Find the line called:

```
wmls.version
```

This allows you to set the property at design time by just adding

```
=1.3.1.1
```

To indicate that your implementation only support 1.3.1.1.

The response to WMLS_GetVersion will return whatever is put as the value to wmls.version.

3.1.2 While running the JAR

The version can be set at runtime by running the server jar with the following argument:

```
--wmls.version=<supported versions(s)>
```

The complete command would look like the following:

```
java -jar server-0.0.1-SNAPSHOT.jar --wmls.version=1.3.1.1,1.4.1.1
```

This will start the server and respond with **1.3.1.1,1.4.1.1** to the WMLS_GetVersion request.

3.2 How to Set Capabilities

This guide documents how to set the server capabilites that the server reports as SUPPORTED by the WMLS_GetCap API.

3.2.1 In Code

The WITSML version can be set in code by modifying the servercap.properties file.

`wmls.contactName`: The name of the person responsible for the server installation `wmls.contactEmail`: The name of the person responsible for the server installation `wmls.contactPhone`: The phone of the person responsible for the server installation

`wmls.changeDetectionPeriod`: The total amount of time elapsed for a change to be detected (required in 1.4.1.1) `wmls.description`: The description of the server `wmls.name`: The name of the server `wmls.supportUomConversion`: 1.4.1.1 only, whether or not the server supports unit of measure conversions `wmls.compressionMethod`: 1.4.1.1 only, the compression method used by the server `wmls.cascadedDelete`: 1.4.1.1 only, whether cascaded deletes are supported

3.3 How to Define Return Messages from GetBaseMsg

This guide documents how to add return messages from GetBaseMsg

3.3.1 In Code

The WITSML version can be set in code by modifying the basemessages.properties file

The file is a list of messages with the structure of:

`basemessage.<returncode>=<message>`

`<returncode>` is the short returned from the operation for which the message is assigned `<message>` is the string message that should be returned for the corresponding short.

3.4 Setting up Logging

This guide documents how to configure logging in Drillflow.

3.4.1 Introduction

Drillflow uses logback-spring.xml to configure logging. The default logging configuration can be seen here on GitHub. here on GitHub.

A tutorial on how to modify this file can be found here: logback.xml Example

This default configuration has 2 appenders, a console appender and a file appender. There are several parts of the file that have been made configurable via environment variables to facilitate running in a container.

If you would like to configure it more deeply (as in adding additional appenders, or changing the logging functionality overall, you will want to skip to externalizing the configuration.

3.4.2 Configurable Properties

In the default logback-spring.xml the following items are configurable via environment variables:

Variable LOGBACK_CONFIG_FILE

Description The path to the logback configuration file to use. This could be mounted from a volume

Default %d %p %C{1.} [%t] %m%n

Example Environmental Switch in Docker To Colorize: -e LOGBACK_CONFIG_FILE='/mnt/config/logback-spring.xml'

Variable CONSOLE_LOGGER_PATTERN

Description The pattern to use when logging to the console.

Default %d %p %C{1.} [%t] %m%n

Example Environmental Switch in Docker To Colorize: -e CONSOLE_LOGGER_PATTERN='%black(%d{ISO8601}) %highlight(%-5level) [%blue(%t)] %yellow(%C{1.}): %msg%n%throwable'

Variable FILE_LOGGER_PATTERN

Description The pattern to use when logging to a file.

Default %d %p %C{1.} [%t] %m%n

Example Environmental Switch in Docker Simplified output: -e CONSOLE_LOGGER_PATTERN='%d{yyyy-MM-dd HH:mm:ss} - %msg%n'

Variable FILE_LOGGER_MAX_SIZE

Default 10MB

Example Environmental Switch in Docker To increase size: -e FILE_LOGGER_MAX_SIZE='20MB'

Variable LOGS

Description The root for a logs directory. It is suggested to log to a mounted volume.

Example Environmental Switch in Docker To increase size: -e FILE_LOGGER_MAX_SIZE='20MB'

Example Docker *run* Command to Set multiple properties:

‘‘ docker run -p 7070:7070 -e VALVE_BASE_URL='https://test.com/' -e VALVE_API_KEY='secret' -e CONSOLE_LOGGER_PATTERN='%black(%d{ISO8601}) %highlight(%-5level) [%blue(%t)] %yellow(%C{1.}): %msg%n%throwable' hashmapinc/drillflow:latest ‘‘

3.4.3 External Configuration

The configuration could be mounted externally via a docker volume that is mapped to the container internally. This is ideal for the case when you would want to centralize a common logging configuration across many containers. You would leverage the *LOGBACK_CONFIG_FILE* as mentioned above.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search